Neural networks
Lab 6: Kohonen networks (self organizing maps)

---

1. Architecture, functioning, training.

The Self-organizing maps (SOM) learn to classify input vectors according to how they are grouped in the input space. They differ from competitive layers in that neighboring neurons in the self-organizing map learn to recognize neighboring regions of the input space. Thus, self-organizing maps learn both the distribution (as do competitive layers) and topology of the input vectors they are trained on.

The neurons in the layer of an SOM are arranged according to a given topology. Neurons which are closed to each other with respect to this topology will represent similar input data.

The functioning of a SOM is similar to that of a competitive network, i.e. for each input data X the winning neuron (that having the weight vector closest to X) is identified.

The training process is also similar to the competitive unsupervised learning (WTA algorithm) but instead of adjusting only the weights corresponding to the winning neuron also the weights of neurons belonging to its neighborhood are adjusted. However the learning rate is smaller as the neuron is at a larger distance from the winning one.

2. Designing Kohonen networks in Matlab.

The function to create a Kohonen network (self-organizing map) is newsom. It can be called as follows:

som=newsom(inputData,[d1,d2,…], topology, distance, learningRate1, epochs1, learningRate2, ns)

Only the first parameter is mandatory. It denotes the set of input data (each data in the set is a column in the matrix inputData). The significance of the other parameters is:

[d1,d2,…]: the size of the functional layer; in the case of a one dimensional functional layer it will be just d1 which will denote the number of neurons; in the case of a two dimensional functional layer the parameter will be [d1 d2]. The implicit value is [5,8].

topology: specifies the type of topology in the case of a two-dimensional layer. The possible values are: 'gridtop' (rectangular structure), 'hextop' (hexagonal structure – the implicit one) and 'randtop' (random structure)

distance: specifies the distance used to define the neighborhood relation on the functional layer. The possible values are: 'dist' (Euclidean distance), 'mandist' (Manhattan distance), 'linkdist' (distance based on the minimal number of edges which should be visited to pass from a neuron to another one – the implicit value).

learningRate1: the learning rate corresponding to the first training stage (the ordering stage which ensures the topological mapping)

epochs1: it denotes the number of epochs corresponding to the first stage (implicit value: 100)

learningRate2: the learning rate corresponding to the second training stage (the fine tuning stage which ensures the adjustment of the prototypes)

ns: the size of the neighborhood during the tuning stage (the implicit value is 1)

A Kohonen network can be trained by using the train function: somt=train(som,data). The total number of epochs can be set as usual by: som.trainParam.epochs=value

One particularity of SOMs is related with the possibility to visualize the way the network map the input data by plotting the weight vectors (when the input data have more than two components then only two components from the weight vectors are plotted. This plot can be realized by using plotsom(somt.IW{1,1},somt.layers{1}.distances).

In the last version of the NN toolbox an interface for training and visualizing Kohonen networks was introduced. It can be called by typing nctool in the command window and it allows the selection of the input data, of the size of the grid (a 2D layer) and the visualization of several elements of the network by using:

plotsompos(somt,data): it visualizes both the input data and the weight vectors (it is similar to the old plotsom)

plotsomnd(somt): it visualizes the neurons, the edges connecting adjacent neurons and illustrates by different colors the distances between the weight vectors corresponding to adjacent neurons (dark colors suggest large distances while light colors suggest small distances)

plotsomhits(somt,data): visualizes the number of input data represented by each neuron

plotsomplanes(somt): visualizes the values of the weights corresponding to each input unit; it allows us to see if there are correlated components (for correlated components the color patterns are similar)

Application 1. *Data analysis with one-dimensional and two-dimensional Kohonen networks*

The input data are generated inside a given domain (e.g. rectangle, circle, ring) and the Kohonen network is trained in order to represent the data by conserving the topological relation between data.

Matlab implementation: som_Data.m

Exercises:

a) Train and visualize a 1D and a 2D Kohonen network to represent data generated inside each domain type
b) Analyze the influence of the grid topology in the case of 2D networks on the quality of the topological mapping for data generated inside a ring
c) Analyze the influence of the number of epochs in the case of a 1D networks on the quality of the topological mapping for data generated inside a ring

Application 3. *Solving the traveling salesman problem (TSP) by using a Kohonen network (elastic net).*

The aim of TSP is to identify the order of visiting a set of towns such that the total length of the route is as small as possible. Using an elastic net one can find an approximate solution of the problem (not necessarily optimal but of acceptable length). The input data are represented by the coordinates of the towns and the aim of the network is to train a one dimensional Kohonen network with a ring topology (the first unit is considered to be a neighbor of the last one and vice versa) to approach the town positions. In order to obtain an acceptable solution the number of units should be at least twice the number of towns. Since the architecture is one dimensional each unit is identified by its position in the networks, i.e. an index, i.

The training process is based on the same idea as in the case of networks used to analyze the data:

$$W^i(t+1) = W^i(t) + \eta(t) \cdot \Lambda(i, i^*) \cdot (X - W^i(t)), \quad \Lambda(i, i^*) = \exp(-\frac{(i-i^*)^2}{2s(t)^2})$$

where the parameter s(t) controls the size of the neighborhood. Usually s has a large enough value in the beginning of the training process (in order to ensure the adjustment of all units) and then is decreased until it becomes small enough (e.g. s=1). A possible rule to decrease s is to divide it by a constant (e.g. 1.25) after every T iterations (e.g. T=100). The learning rate, $\eta(t)$, is also decreasing (e.g. $\eta(t+1)=0.995\ \eta(t)$).

Matlab implementation: ElasticNet_TSP.m

Exercises:

a) Compute the approximate length of the final tour (this is the sum of the distances between the weight vectors corresponding to adjacent units).

b) Analyze the influence of the initialization of the weights on the quality of the obtained tour after a given number of iterations. Two variants: (i) weights randomly initialized in [a,b] x[a,b]; (ii) weights initialized on a circle having the center in ((a+b)/2,(a+b)/2) and a radius equal to (b-a)/4.

c) Analyze the influence of the method of adjusting the parameter eta on the quality of the obtained tour after a given number of generations. Variants: (i) eta(t+1)=eta(t)*0.995; (ii) eta(t+1)=eta(0)/log(t+1));

d) Analyze the influence of the method of adjusting the parameter s on the quality of the obtained tour after a given number of generations. Variants: different values for the factor used to divide s (e.g. 1.25, 1.5, 2, 0.75).